Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays

Robert Brummayer and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria
{robert.brummayer,armin.biere}@jku.at

Abstract. Satisfiability Modulo Theories (SMT) is the problem of deciding satisfiability of a logical formula, expressed in a combination of first-order theories. We present the architecture and selected features of Boolector, which is an efficient SMT solver for the quantifier-free theories of bit-vectors and arrays. It uses term rewriting, bit-blasting to handle bit-vectors, and lemmas on demand for arrays.

1 Introduction

A new kind of verification engines, called Satisfiability Modulo Theories (SMT) solvers, gained a lot of interest both in research and industry recently. SMT generalizes pure boolean satisfiability (SAT) and provides first-order theories to express design and verification conditions of interest. For example, important first-order theories are fixed-size bit-vectors, arrays, linear arithmetic, and difference logic. An SMT solver takes a formula expressed in a combination of first-order theories as input, and decides satisfiability. Additionally, if the instance is satisfiable, then most SMT solvers provide a model. For more information about SMT and first-order theories see for example [4,10,12].

Boolector is an efficient SMT solver for the quantifier-free theory of bit-vectors in combination with the extensional theory of arrays. Bit-vectors can be used to express designs and specifications directly on the word-level, while arrays can be used to model memory, e.g. main memory in software, or memory components like caches and FIFOs in hardware systems. The combination of bit-vectors and arrays allows reasoning about software and hardware on a more appropriate level than pure propositonal logic. Generally, SMT solvers benefit from the additional word-level information and generates word-level models. Boolector provides concrete models for bit-vector *and* arrays.

The SMT competition [2] in 2008 showed, that there has been a lot of progress in the SMT community. In each division the winner clearly outperformed last year's winner. In particular, the performance of SMT solvers in the quantifier-free theory of bit-vectors `QF_BV`, and bit-vectors with arrays and uninterpreted functions `QF_AUFBV`, increased heavily. Boolector entered the SMT competition for the first time. It participated in exactly these two divisions and won both. In `QF_BV` it solved 18 formulas more than Z3.2 and 92 formulas more than last year's winner Spear v1.9 [1]. In QF_AUFBV Boolector solved 16 formulas more than Z3.2 and 64 more than last year's winner Z3 0.1 [8].

2 Architecture

Boolector depends on term rewriting and bit-blasting for bit-vectors. Lemmas on demand [9] are used to handle the extensional theory of arrays lazily [5]. It takes a formula expressed in the SMT-LIB format [11], or alternatively in the BTOR format [6], as input. The BTOR format is a low-level bit-vector format with clean semantics that is easy to parse. Additionally, BTOR supports bit-vector arrays and model checking of safety properties. In addition to these input formats, Boolector also provides a rich C API, which allows to use Boolector as library. Boolector is implemented in C. A schematic overview is shown in Fig. 1.

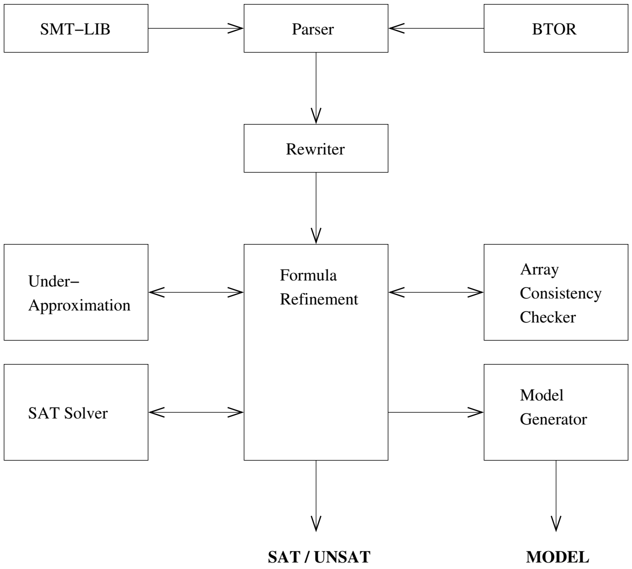

Fig. 1. Schematic overview of Boolector.

Parser The parser reads the input and builds an abstract syntax DAG. During the parse process, basic rewriting rules are applied to simplify the DAG.

Rewriter The rewriting engine provides rewrite rules that can be divided into three levels. By default, all rewrite levels are applied to simplify the input. Level 1 rewrite rules are basic rules, e.g. $a \wedge \neg a \Leftrightarrow \bot$, that are applied during formula construction. Level 2 rewrite rules consist of global term substitutions in combination with static analysis techniques. Before terms are substituted, they are topologically sorted. Then, term substitutions are performed in one pass. Level 3 rewrite rules perform arithmetic normalization. Rewrite rules of quadratic worst case complexity are additionally bounded in recursion depth.

Array Consistency Checker This checker is one main component in the lemmas on demand approach for the extensional theory of arrays [5]. It checks if the current assignment by the SAT solver is consistent with the theory.

Under-approximation Boolector supports under-approximation of bit-vector variables and reads on arrays. This module is responsible for realizing under-approximation directly on the CNF level. Under-approximation constraints are added as clauses that additionally constrain the search space.

SAT Solver PicoSAT [3] is used as SAT solver. It uses state of the art techniques like watching literals, phase saving, conflict learning and rapid restarts. Boolector uses PicoSAT incrementally to decide the extensional theory of arrays, and for under-approximation.

Model Generator The ability to provide concrete models in the satisfiable case cannot be overestimated. In generaral, a satisfiable formula corresponds to a bug that has been found. The ability to provide a concrete counter example that can be directly used for debugging is one of the main features in the success story of model checking [7]. Boolector can output concrete bit-vector *and* array models. The array models are (partially) instantiated arrays, where indices and values are concrete bit-vectors.

Formula Refinement The formula refinement is the heart of Boolector. Initially, the bit-vector part is translated to SAT while the array part is abstracted with the help of fresh bit-vector variables. The refinement loop calls the SAT solver to obtain an assignment. If the result is unsatisfiable, the loop terminates with *unsatisfiable*. However, if the result is satisfiable, the array consistency checker is used to check if the current assignment is consistent with the extensional theory of arrays. If the current assignment is consistent, then the formula is *satisfiable*. However, if the assignment is inconsistent, a lemma on demand, that rules out this and similar assignments, is added as as formula refinement. Additionally, under-approximation refinement can be enabled for bit-vector variables and reads. In this case the under-approximation refinement and the lemmas on demand refinement for the extensional theory of arrays are intertwined.

3 Selected Features

Model Checking Boolector can also be used as incremental model checker for word-level safety properties of synchronous hardware systems with memories [6]. The BTOR format provides a sequential ”next“ operator, which can be used to express state transitions of bit-vector registers and memories. Boolector supports bounded model checking for witnesses, and k-induction with and without all-different constraints. All-different constraints are used for simple paths.

Under-approximation Boolector supports several under-approximation techniques and refinement strategies. Bit-vector variables and reads on arrays can be additionally constrained on the CNF level. The under-approximation can be refined locally or globally. In the global refinement strategy, the under-approximation is refined equally for all variables and reads. In the local strategy the under-approximation refinement is individually performed for each variable and read. The under-approximation feature allows to generate ”smaller“ models that are typically easier to interpret by users.

Pretty Printer Boolector allows to convert formulas from BTOR to SMT-LIB format and vice versa. The pretty printer can be combined with Boolector's rewriting module to internally simplify the formula before conversion.

4 Conclusion

We presented Boolector, which is an efficient SMT solver for the quantifier-free theories of bit-vectors and arrays. Boolector uses term rewriting, bit-blasting for bit-vectors, and lemmas on demand for arrays. We discussed its architecture, main concepts, and selected features.

References

1. D. Babic. *Exploiting Structure for Scalable Software Verification.* PhD thesis, 2008.
2. C. Barrett, M. Deters, A. Oliveras, and A. Stump. SMT-Comp, 2008. www.smtcomp.org.
3. A. Biere. PicoSAT essentials. *JSAT*, 4, 2008.
4. A.R. Bradley and Z. Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer, 2007.
5. R. Brummayer and A. Biere. Lemmas on demand for the extensional theory of arrays. In *Proc. SMT*, 2008.
6. R. Brummayer, A. Biere, and F. Lonsing. BTOR: Bit-precise modelling of word-level problems for model checking. In *Proc. BPR*, 2008.
7. E.M. Clarke. The birth of model checking. In *25 Years of Model Checking*, 2008.
8. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. TACACS*, 2008.
9. L. de Moura and H. Rueß. Lemmas on demand for satisfiability solvers. In *Proc. SAT*, 2002.
10. D. Kroening and O. Strichman. *Decision Procedures: An algorithmic Point of View.* Springer, 2008.
11. S. Ranise and C. Tinelli. The satisfiability modulo theories library (SMT-LIB), 2008. www.smt-lib.org.
12. R. Sebastiani. Lazy satisfiability modulo theories. *JSAT*, 3, 2007.